ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Realisierung einer kontextsensitiven mobilen Anwendung für therapeutische Hausaufgaben

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Niklas Häusele
niklas.haeusele@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2017

Fassung 12. Dezember 2017

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Diese Arbeit beschäftigt sich mit der Konzeption und Realisierung einer mobilen Anwendung, die unter verschiedenen Kontexten therapeutische Hausaufgaben unterstützt. Dabei wird ein flexibles Modell entwickelt, sodass der Prototyp viele verschiedene Anwendungen haben kann. Hierbei wird eine mobile Anwendung entwickelt, welche auf die vielen verschiedenen therapeutischen Hausaufgaben angepasst werden kann. Es wurde ein Triggerkonzept entwickelt, mit dem variabel auf Kontexte durch unterschiedliche Übungen reagiert werden kann. Für die Realisierung des Prototypen wurde das Ionic-Framework verwendet um damit eine hybride mobile Anwendung zu entwickeln.

Das Ziel der Arbeit ist aufzuzeigen, wie ein flexibles Modell aussehen kann, mit dem therapeutische Hausaufgaben unter verschiedenen Kontexten gestartet werden können. In der Arbeit werden außerdem die unterschiedlichen Technologien die zum Einsatz gekommen sind, besprochen. Es wird außerdem ein User-Interface und Bedienkonzept implementiert, das für eine solche Anwendung sinnvoll wäre. Der Beitrag der Bachelorarbeit besteht damit in dem Ausprobieren verschiedener Konzepte, um darüber zu urteilen, welche Konzepte in einer produktionsreifen mobilen Anwendung zum Einsatz kommen sollten.

Danksagung

Ich danke meinem Betreuer Marc Schickler für die sehr angenehme Betreuung. Außerdem danke ich meiner Familie, Freunden, Kommilitonen und Arbeitgebern, die mich während meines Studiums immer unterstützt und motiviert haben.

Inhaltsverzeichnis

1

Einleitung

Therapeutische Hausaufgaben und Interventionen sind heutzutage ein viel genutzes Mittel für Therapeuten bei der Behandlung von Patienten. Die Wirksamkeit therapeutischer Hausaufgaben ist bereits durch viele verschiedene Studien belegt worden [1]. Mit der immer wachsenden Zahl an modernen mobilen Endgeräten stellt sich die Frage, inwieweit mobile Endgeräte unterstützend sein können. Aufgrund der enormen Vielfalt an verschiedenen Hausaufgabenarten, Kontexten unter denen die Bearbeitung stattfinden kann und Aufgabentypen ist es schwierig, eine Anwendung zu entwickeln, die flexibel genug ist um mehrere Behandlungstypen und Hausaufgabenarten abzudecken. Meist müssen für spezifische Behandlungen spezifische Anwendungen entwickelt werden. Durch die bei der Entwicklung entstehenden Kosten hat sich die Behandlung begleitend mit therapeutischen Hausaufgaben mit *smart-mobile Devices* noch nicht durchgesetzt. Weiterhin sind Entwicklungszyklen sehr lange, was es schwierig macht, den Bedürfnissen des Therapeuten und des Patienten zu genügen. Diese Arbeit beschäftigt sich also mit einem Prototypen, an dem aufgezeigt werden kann, wie man eine generische Smartphone Anwendung bauen könnte, die durch den Therapeuten einfach konfigurierbar und flexibel genug ist, um verschiedene Kontexte bei der Ausübung von Hausaufgaben und verschiedene Aufgabenkonfigurationen bereitzustellen. Diese Konzepte können dann in eine finale Version der Anwendung übernommen werden, um damit eine generische Anwendung zu bauen, die viele verschiedene Fälle abdeckt und eine Behandlung mit therapeutischen Hausaufgaben mit mobilen Geräten der breiten Masse zugänglich macht.

1.1 Ziel der Arbeit

Als Ziel der Arbeit soll ein Prototyp entworfen werden, der eine Übungsausführung startet je nach Kontext in dem sich der Patient derzeit befindet. Durch den zur Arbeit entwickelten Prototypen soll eine mögliche Implementation, vorallem für den Aspekt des Auslösens von therapeutischen Hausaufgaben, aufgezeigt werden. Auch soll der Prototyp ein entsprechendes Bedienkonzept aufweisen. Es sollte außerdem eine mögliche User-Interface Gestaltung ausprobiert werden. Neben den Grundlagen der verwendeten Technologien, werden die Vor- und Nachteile bei der hybriden App-Entwicklung mit Ionic besprochen für eine Anwendung, die Patienten bei der Übungsausführung unterstützen soll. Die Arbeit kann damit für eine Umsetzung als Leitfaden verwendet werden, wenn es darum geht ein Bedienkonzept zu entwickeln, sowie ein Modell für die kontextbasierte Übungsausführung. Dadurch wäre eine finale Implementation mit weniger Aufwand und bereits erprobten Konzepten schneller umzusetzen.

1.2 Aufbau der Arbeit

In Kapitel 2 beschäftigt sich diese Arbeit zunächst mit den Grundlagen therapeutischer Hausaufgaben. Dabei werden die Definition, Ziele, Arten und der Einsatz von mobilen Endgeräten bei dem Einsatz von Hausaufgaben vorgestellt. Im darauffolgendem Kapitel 3 werden die funktionalen und nicht-funktionalen Anforderungen an den entwickelten Prototypen aufgelistet. In Kapitel 4 werden die wichtigsten Technologien, die bei der Entwicklung zum Einsatz gekommen sind, erläutert. Im nächsten Kapitel 5 wird der Entwurf mit den wichtigsten Komponenten vorgestellt und das entwickelte Triggerkonzept erläutert. Als nächstes werden in Kapitel 6 die Anforderungen mit dem tatsächlich entwickelten Prototypen abgeglichen. Zuletzt folgt in Kapitel 7 eine Zusammenfassung der Ergebnisse sowie ein Fazit und ein Ausblick, der weitere Ideen für eine erfolgreiche Umsetzung beschreibt.

2

Grundlagen therapeutischer Hausaufgaben und Interventionen

Dieses Kapitel stellt die Grundlagen therapeutischer Hausaufgaben und Interventionen dar. Therapeutische Hausaufgaben werden nach Borgart und Kemmler dadurch beschrieben, dass Sie nicht im Behandlungszimmer stattfinden, sondern außerhalb des Behandlungszimmers. Die konkreten Hausaufgaben können dabei sehr unterschiedliche Aufgaben beinhalten [2]. Ein Therapeut könnte dem Patient beauftragen, unter bestimmten äußerlichen Umständen eine Übung durchzuführen. Ein Psychotherapeut könnte beispielsweise eine kurze Atempause nach dem Eintreten von Platzangst verordnen. Zu therapeutischen Hausaufgaben kann dabei auch das Protokollieren von Uhrzeit, Ort und das Beantworten von dem Therapeuten gegebenen Fragen gehören.
Für den Therapeuten gibt es verschiedene Gründe solche Hausaufgaben zu erteilen. Studien haben gezeigt, dass zum Beispiel in der kognitiven Verhaltenstherapie das Erteilen von therapeutischen Hausaufgaben den Behandlungsverlauf verbessert hat, im Vergleich zu Behandlungen in denen keine Hausaufgaben erteilt wurden [3]. Es gibt eine Vielzahl von möglichen Aufgaben, die ein Therapeut stellen kann. Neben der eigentlichen erteilten Hausaufgabe ist für den Therapeuten auch interessant, zu welchem Grad der Patient sich an die vereinbarten Konditionen gehalten hat oder unter welchen Umständen er die Hausaufgabe erledigt.

2.1 Definition von Hausaufgaben

Dieses Kapitel definiert therapeutische Hausaufgaben näher. In der heutigen Zeit werden als eine häufige Interventionsart Hausaufgaben für einen Patient erteilt [4]. Laut Borgart und Kemmler sind Hausaufgaben dadurch definiert, dass Sie außerhalb des Therapiezimmers und zeitlich zwischen den einzelnen Therapiesitzungen erledigt werden sollten. Die Art und die konkrete Aufgabe können dabei sehr weit gefächert sein. Das heißt, es sind viele verschiedene Aufgaben denkbar, von dem Aufschreiben von Notizen, Entspannungsübungen bis hin zu sportlichen Aufgaben. Ein weiteres wichtiges Prinzip bei therapeutischen Hausaufgaben ist die Adhärenz. Adhärenz ist das Maß, in dem beurteilt wird, wie genau sich der Patient an die Abmachungen für die Durchführung der Hausaufgaben gehalten hat. So ist es für den Therapeuten wichtig, möglichst genau zu wissen, welchen Grad an Adhärenz der Patient bei der Ausführung der Hausaufgabe hatte. Durch eine geringe Adhärenz könnte es zu dem logischen Schluss für den Therapeuten kommen, die Hausaufgabe zu ändern, obwohl sie unter Umständen lediglich falsch ausgeführt wurde. Auch das Erkennen einer absichtlich falschen Durchführung oder gar keiner Durchführung hilft einem Therapeuten bei der weiteren Behandlung. Man sieht daran deutlich, wie wichtig es ist, dass der Therapeut eine Vorstellung über die Adhärenz des Patienten hat, da es sonst zu einer starken Beeinträchtigung der Behandlung kommen kann [2].

2.2 Ziele von Hausaufgaben

Bevor sich ein Therapeut über das Stellen von Hausaufgaben Gedanken macht, muss er sich im Klaren sein, was für Ziele er mit den Hausaufgaben verfolgt. Gestellte Aufgaben sollten also daran ausgerichtet sein, welches Ziel mit ihnen beabsichtigt ist. Nach Wendlandt, Freeman und Rosenfield gibt es viele verschiedene Ziele, die im Folgenden aufgelistet und kurz erläutert werden [2].

- Üben von neuen erlernten Verhaltens- und Denkweisen
- Neue Erkenntnisse durch Selbstbeobachtung über sich selbst erlangen

- Therapieinhalte festigen
- Therapieinhalte stärker in den Alltag integrieren
- Motivation für die Behandlung steigern
- Neue Denk- und Verhaltensweisen in den Alltag aufnehmen
- Eine Effizienzsteigerung der Therapie

2.3 Arten von Hausaufgaben

Dieses Kapitel beschäftigt sich mit den verschiedenen Arten von Hausaufgaben und einer Systematisierung der verschiedenen Hausaufgaben. Als Beispiel könnte eine Aufgabenstellung von einem Therapeuten wie folgt lauten: „Immer wenn du Schmerzen im unteren Rücken bekommst, musst du 60 Sekunden lang aufstehen und dabei deine Arme kreisen.“ Dabei stellt die Bedingung, dass man die Übung ausführen soll sobald man Rückenschmerzen hat, den Kontext dar. Ein Kontext bei psychotherapeutischer Behandlung könnte zum Beispiel sein: „Sobald du Platzangst bekommst“. So einen Kontext könnte man zum Beispiel messen über einen Pulsmesser, der anhand des Pulses und Schweisses auf der Haut die Anwendung darüber informiert, und die Anwendung abwägen kann, ob sie eine Übung startet. Wie man an diesem Beispiel sehen kann, ist die konkrete Hausaufgabe sehr stark davon abhängig, welcher Art von Behandlung der Patient unterstellt ist. So hat ein Physiotherapeut Aktivitätsübungen für zum Beispiel Muskelstärkung, während ein Psychotherapeut Reflexionsaufgaben stellen könnte. In einem unveröffentlichen Manuskript von Breil werden Hausaufgaben nach Kategorien eingeteilt und sortiert [2]. Diese sind in Abbildung 2.1 graphisch sortiert dargestellt und entsprechende Beispielaufgaben mit einsortiert.

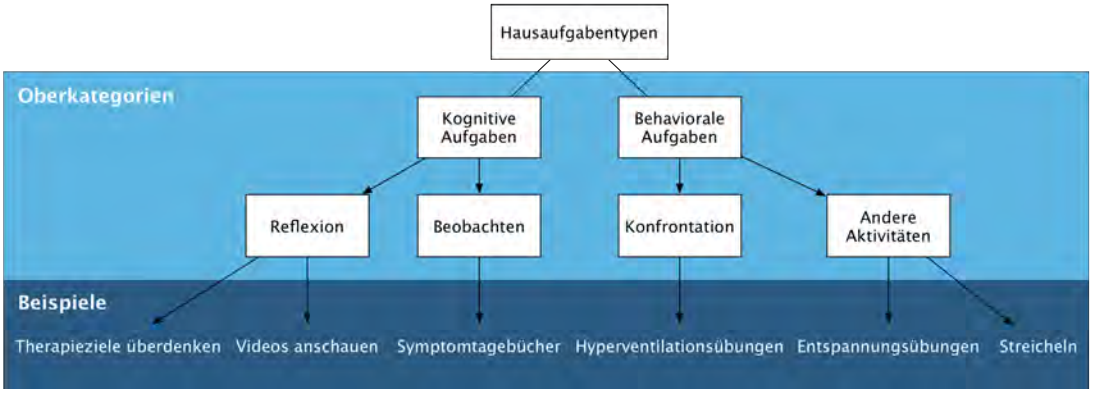

Abbildung 2.1: Beispiele für Aufgaben und Kategorien [2]

2.4 Mobile Endgeräte bei dem Einsatz von Hausaufgaben

Auch Therapeuten haben heute das Verlangen nach dem modernen Einsatz von Smartphones im Behandlungsalltag [5]. Jedoch gibt es derzeit verschiedene Probleme mit vorhandenen mobilen Anwendungen. So sind diese oft nicht flexibel genug verschiedene Hausaufgaben abzubilden. Weiterhin ignorieren diese spezifischen Entwicklungen oft den Kontext in dem die Hausaufgaben ausgeführt werden und es gibt keine Möglichkeit für eine Art des Feedbacks nach Erledigung der Hausaufgabe für den Patienten. Außerdem werden Features von modernen Smartphones oft ignoriert, obwohl diese die Behandlung deutlich verbessern könnten. So können Smartphones zum Beispiel zur Unterstützung von Entspannungsübungen Musik abspielen, Sensordaten aufzeichnen und diese dann dem Therapeuten zur Verfügung stellen. Neben dem Therapeuten sind solche aufgezeichneten Daten auch interessant für die Wissenschaft, die mit den gesammelten Daten Studien machen kann und dadurch neue Erkenntnisse gewinnen kann. Durch diese neu gewonnenen Erkenntnisse profitieren dann wiederum die Patienten, da diese eine effektivere Behandlung von dem Therapeuten erhalten können. [5]. Eine moderne technologische Lösung für therapeutische Hausaufgaben ist damit unabdingbar. Der Patient erhält eine stärker personalisiertere Behandlung, die auch von dem Therapeuten jederzeit angepasst werden kann. Dies könnte auch über eine Weboberfläche für den Therapeuten geschehen, dadurch wird auch der Aufwand für

das Ändern der Behandlung minimiert [1]. Mögliche beispielhafte Teile einer solchen Weboberfläche werden in 2.2 dargestellt.

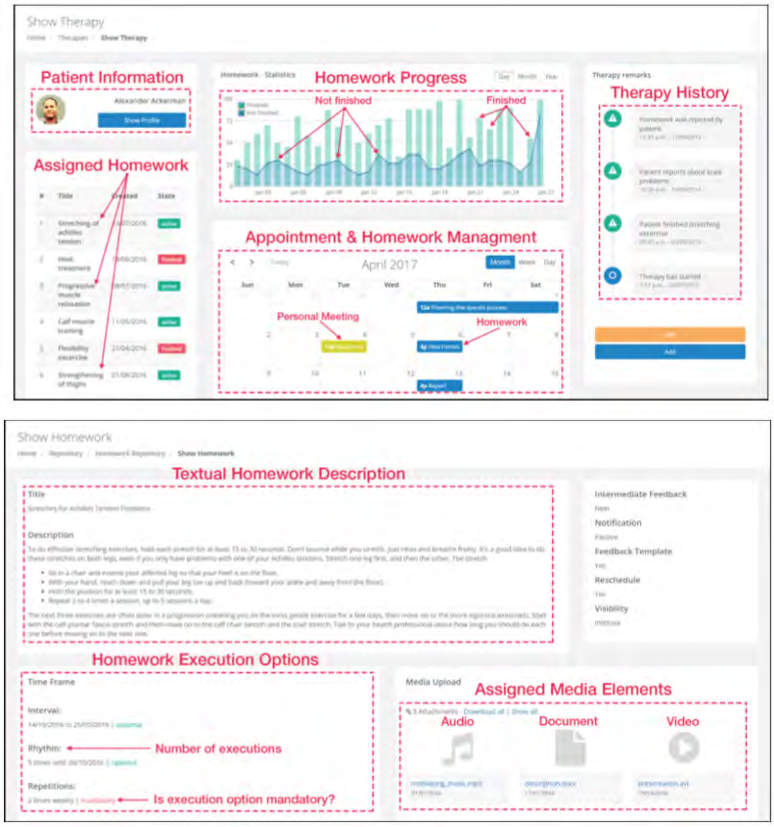

Abbildung 2.2: Beispiel für Teile einer Weboberfläche für den Therapeuten [5]

Weiterhin kann damit die Zeit, in der der Patient in der Behandlung ist, effektiver genutzt werden. Durch eine Anwendung mit entsprechender Erklärung der Aufgabe, kann die Zeit in der Behandlungsstunde für Sachen genutzt werden, die nicht über technologische Anwendungen vorerst gelöst werden können, wie zum Beispiel mehr Zeit für ein persönliches, vertrautes Gespräch.

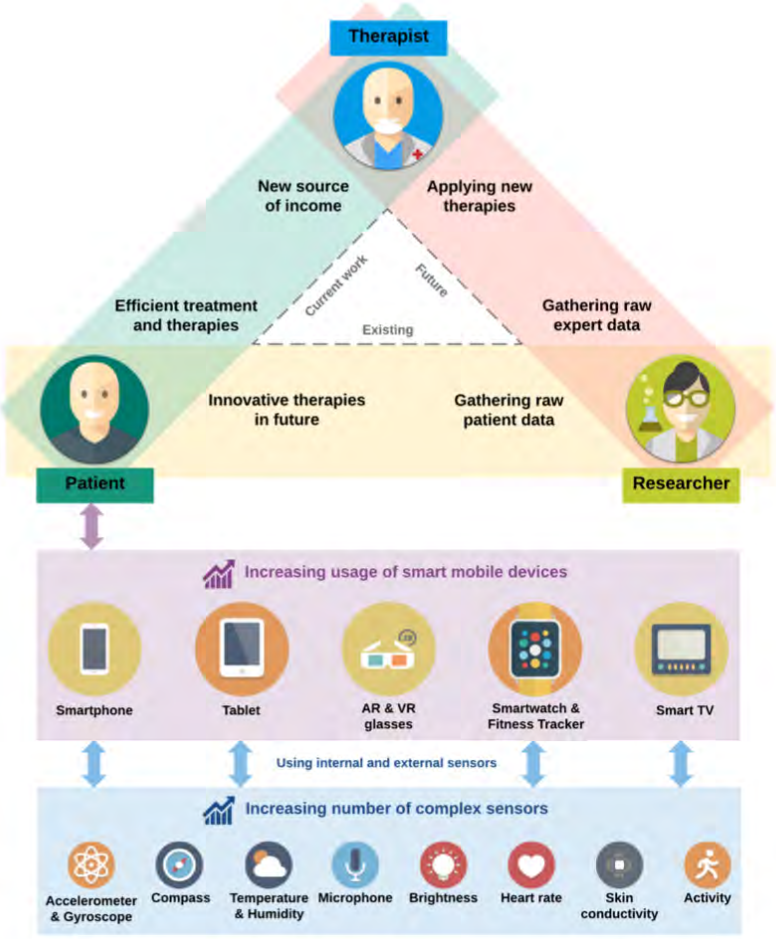

Abbildung 2.3: Überblick von Hausaufgaben mit mobilen Endgeräten [1]

Wichtig für eine erfolgreiche Umsetzung von Hausaufgaben auf mobilen Endergeräten ist außerdem, dass eine Lösung nicht nur auf dem mobilen Gerät flexibel ist, sondern auch in den anderen Phasen und Anwendungen eines modernen Ansatzes. Dabei ist eine Flexibilität zur Designzeit eines Hausaufgabenprozess unerlässlich, die eben eine Vielfalt an Möglichkeiten bietet. Neben einer Designflexibilität ist eine *Runtime* Flexibilität wichtig. Diese muss während der Ausführung die benötigten Komponenten für eine erfolgreiche Durchführung des Hausaufgabenprozesses laden [4].

3

Anforderungen

3.1 Funktionale Anforderungen

Dieses Kapitel listet die funktionalen Anforderungen an den Prototypen auf. Diese werden der Wichtigkeit nach bewertet mit einer $++$, $+$ und 0 Symbolik.

F1 Starten von Übungen ++

Die Applikation muss in der Lage sein, basierend auf verschiedenen Kontexten eine Übungsausführung zu starten. Die Kontexte sollten dabei auf Gerätesensoren hören und Features eines modernen Smartphones ausnutzen.

F2 Nutzen von Gerätesensoren ++

Die Anwendung sollte verdeutlichen, wie Sensoren und Funktionen von *smart-mobile Devices* eine Übungsausführung starten können. Dabei nutzt sie zum Beispiel ein eingebautes Pedometer.

F3 Konfigurieren von Übungen ++

Je nach Konfiguration muss die Anwendung in der Lage sein, je nach Kontext eine unterschiedliche Übung zu starten. So soll es möglich sein mehrere Kontexte und Übungen zu beachten bzw. zu starten.

F4 Erklärung von Übungen ++

Sobald eine Übung gestartet wird, muss der Patient eine Erklärung für die Übung erhalten. Diese sollte konfigurierbar sein. Die Erklärungsblöcke sollen dabei unterschiedlich angeordnet werden können.

F5 Konfiguration von selbstauslösenden Übungen +

Der Therapeut hat die Möglichkeit den Homescreen für den Patienten passend zu den besprochenen Übungen zu konfigurieren. Dafür soll er die Farbverläufe und Icons ändern können.

F6 Mediaelemente in der Übungserklärung +

Übungserklärungen sollten durch verschiedene multimediale Elemente eine benutzerfreundliche und leicht verständliche Übersicht über die auszuführende Übung geben. Dazu gehören das Abspielen einer Audiodatei und das Abspielen eines Videos.

F7 Bestätigen einer Übungserklärung +

Der Patient sollte nach erfolgreicher Übungserklärung wissentlich die Übungsausführung starten und damit zur Kenntnis geben, dass die Erklärung erfolgreich verstanden wurde.

F8 iBeacon-basierte Ausführung +

Ein Kontext unter dem eine Übung starten soll, ist das Betreten eines Bereichs mit einem iBeacon. Dabei sollte der iBeacon, auf den das Gerät achten soll, konfigurierbar sein.

F9 Repetitionszähler +

Eine Übungsart soll das Wiederholen von Aktionen sein, bei der der Patient mit einem Zähler eine bestimmte Wiederholungszahl erreichen muss bevor die Übung vorbei ist. Diese Wiederholungszahl soll konfigurierbar sein.

F10 Fragebogen +

Es soll ein Fragebogen nach dem Ausführen einer Übung dargestellt werden. Dieser sammelt nach einer Übung über verschiedene Formularelemente Informationen über Patienten. Dieser soll durch Knopfdruck die gesammelten Daten abschicken können.

F11 Grafikantwort +

Eine mögliche Frage soll ein Zeichenfeld sein, indem der Patient mit dem Finger eine Zeichnung oder eine Unterschrift zeichnen kann. Dabei sollte als Antwort das Bild der Zeichnung übermittelt werden.

F12 Stimmungsslider +

Eine weitere Art eine Frage zu beantworten soll ein Slider sein, mit dem der Patient seine Stimmung nach dem Ausführen der Übung darstellen kann. Dieser soll horizontal mit einem Knopf verschiebbar sein.

F13 Kameraantwort +

Ein weiteres Eingabefeld soll das Öffnen der Gerätekamera sein. Diese ermöglicht dann als Antwortmöglichkeit ein Bild aufzunehmen. Als Antwort soll das geschossene Bild übermittelt werden. Außerdem soll eine Vorschau des Bildes angezeigt werden.

3.2 Nicht-funktionale Anforderungen

Dieses Kapitel listet die nicht-funktionalen Anforderungen an den Prototypen auf. Diese werden der Wichtigkeit nach bewertet mit einer $++$, $+$ und 0 Symbolik.

NF1 Erweiterbarkeit +

Da der Fokus des Prototypen auf Erweiterbarkeit liegt, sollten die Kontexte unter denen Übungen starten können leicht erweiterbar sein.

NF2 Konfiguration zur Laufzeit +

Die verschiedenen Übungen und Kontexte, die für den Patienten wichtig sind, sollten ohne ein erneutes Installieren der App konfigurierbar sein.

NF3 Interface ++

Das Interface sollte optisch ansprechend sein, damit der Patient die Anwendung leicht und ohne viel Einlernzeit bedienen kann.

NF4 Usability ++

Die Anwendung sollte eine hohe Usability aufweisen und es vermeiden, dass der Benutzer unabsichtliche Fehler begehen kann.

NF5 Performance +

Die Anwendung sollte eine Performance haben, die den Patient nicht daran hindert die Übung korrekt auszuführen oder auf die Anwendung für eine Übung zu warten.

NF6 Offline-Verfügbarkeit 0

Die Applikation sollte auch ohne eine bestehende Internetverbindung eine Übungsausführung ermöglichen.

4

Technologien

4.1 Grundlagen hybrider App-Entwicklung

Der Begriff *hybride App-Entwicklung* wird verwendet für die Entwicklung von Applikationen, die eine Mischung aus Webanwendung und nativer Anwendung sind. Im allgemeinen Sprachgebrauch wird oft der Begriff *hybride Entwicklung* mit dem Begriff der *Cross-Plattform-Entwicklung* vermischt. Die Cross-Plattform Entwicklung ist dabei die Entwicklung mit einer Codebasis für mehrere Softwareplattformen gleichzeitig. Ein mögliches Szenario hierbei wäre eine Applikation, die mit der gleichen Codebasis eine Android sowie eine IOS Applikation erzeugt, ohne plattformspezifischen Code schreiben zu müssen. Weiterhin hat sich das Apache Cordova Framework als defacto Standard für hybride Entwicklung etabliert. Deswegen wird auch oft implizit angenommen, dass hybride Entwicklung immer bedeutet, dass Apache Cordova verwendet wird. Im Zentrum steht jedoch, dass die hybride App-Entwicklung sich für den Großteil der Anwendungsfunktionalität der jeweiligen Webview des Host OSes bedient. Webviews sind in nativer App Entwicklung die Bezeichnung für Komponenten, die dem Programmierer Zugriff auf eine Browserimplementation bieten und es ermöglichen interaktiven Web-Content anzuzeigen. Apache Cordova ermöglicht es Apps in der Webview Zugriff auf native Funktionen zu gewähren. Dies geschieht durch eine native Schnittstelle, die ohne Apache Cordova nicht für den Browser möglich wäre. Mit der Definition der wichtigsten Begriffe kann nun in den nächsten Kapiteln die verwendeten Frameworks und Techniken erklärt werden.

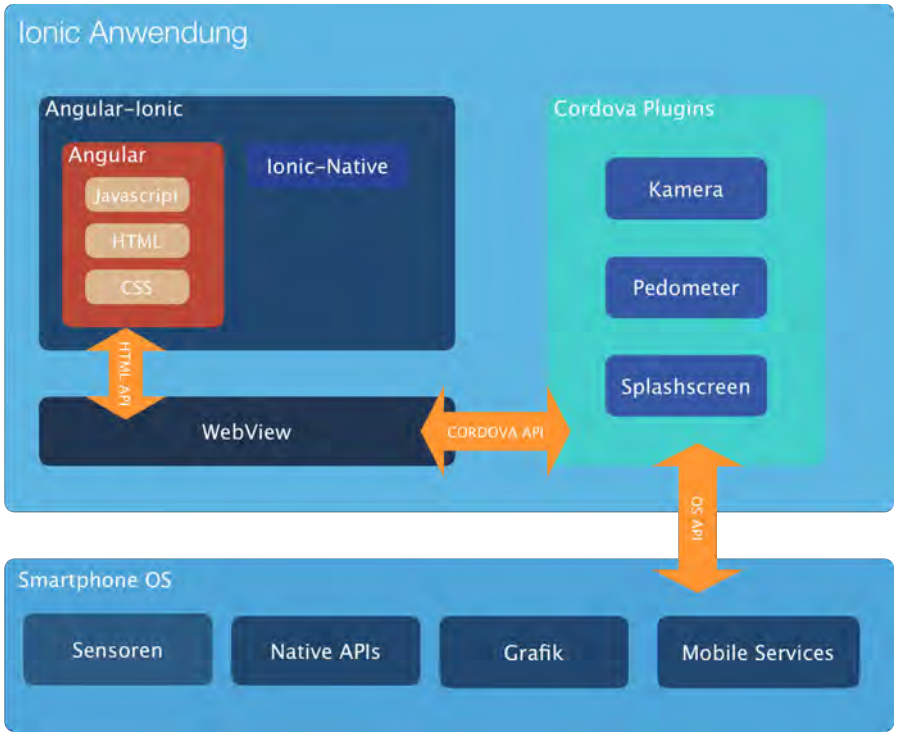

Abbildung 4.1: Ionic-Angular Architektur

4.1.1 Apache Cordova

Wie in Grundlagen hybrider App-Entwicklung angedeutet, ist Apache Cordova ein Framework, das eine Entwicklung von hybriden cross-plattform Anwendungen ermöglicht. Das Projekt steht unter der Apache Lizenz 2.0 [6]. Eine Besonderheit von Cordova ist, dass mit einer Codebasis viele verschiedene Plattformen unterstützt werden können. Man kann einen Großteil des Codes in Javascript schreiben und native Funktionalitäten der jeweiligen Plattform verwenden. Dabei unterstützt Cordova alle großen Plattformen wie Android, iOS, macOS, Windows und Windows Phone. Weiterhin bietet Cordova auch ein Command-Line Interface mit dem Projekte erstellt und verwaltet werden können.

Die typische Cordova App besteht aus verschiedenen Bestandteilen, die im Folgenden kurz erläutert werden. Die Besonderheit einer Cordova Anwendung ist die HTML Rendering Engine. Diese stellt im Gegensatz zu einer nativen UI, HTML mit CSS und Javascript dar, genau wie mit einem üblichen Internet Browser. Cordova Plugins sind die

Schnittstelle, die Zugriff auf native OS APIs haben. Sie stellen die Brücke dar zwischen Web API und nativer API. So kann ein Plugin Zugriff auf das im Smartphone verbaute Accelerometer haben und die ausgelesenen Daten über eine Javascript Implementation der Web App zur Verfügung stellen. Dafür ist es nicht notwendig, dass der Web-Browser über eine native Accelerometer HTML5-Implementation verfügt. Plugins sind dabei so aufgebaut, dass Sie falls möglich für alle Plattformen den plattformspezifischen Code implementieren und dann eine einzige Javascript API bieten. So kann man zum Beispiel den Zugriff auf eine Gerätekamera erhalten, ohne für iOS Swift Code oder für Android Java Code zu schreiben, sondern man kann eine gemeinsame Javascript API verwenden. Je nach Plattform wird dann der nötige native Brückencode verwendet. Ein weiterer Teil ist die WebView in Cordova. So ist das Konzept hinter den meisten Cordova Apps eine WebView, die sich über die komplette Oberfläche erstreckt. Die WebView die den kompletten Bildschirm füllt, zeigt nun die Webanwendung an, die bei einer Cordova Anwendung entwickelt wird. Nun kann die komplette App im Browser entwickelt werden, mit HTML, CSS und Javascript. Diese Architektur wird in der Grafik 4.1 noch einmal bildlich dargestellt [7]. Bei der Distribution der Cordova Anwendung werden alle benötigten Plugins, der benötigte native Code für eine WebView, die Web-App und Resource Dateien erstellt und in eine Anwendung zusammengefasst, die dann auf dem jeweiligen Gerät als native App ausgeführt werden kann [7].

4.1.2 Angular-Ionic

Dieses Kapitel beschreibt, inwieweit Angular und Ionic zusammenhängen und inwieweit sie sich unterscheiden. Wie in Kapitel 4.4 erklärt, ist Angular ein komponentenbasiertes Single-Page Application Framework. Ionic bietet durch eine große Komponentenlibrary Zugriff auf viele fertige Angular Komponenten, die nativen App Elementen gleichkommen. So kann zum Beispiel mit Ionic ein Tablayout realisiert werden, indem die fertigen Angular Komponenten von Ionic verwendet werden, ohne auf die nativen UI Libraries zurückgreifen zu müssen. Weiterhin bietet Ionic verschiedene Angular Directives und Services, die bei der mobilen Anwendungsentwicklung routinemäßig gebraucht werden, so zum Beispiel das Detektieren der Version des Betriebssystems oder das Darstellen

von Elementen abhängig von dem Betriebssystem. Ein Teil in dem sich Ionic vollständig von Angular unterscheidet, ist das Routing der Applikation. Während Angular einen `Router` verwendet [8], verwendet Ionic hingegen einen `NavController` für die App-Navigation [9]. Letzterer ist besser angepasst auf eine *app-like* Navigation. So wird mit dem `NavController` eine iOS ähnlicher Navigationsstack verschiedener Pages aufgebaut. Mit diesem `NavController` kann ein App Seitenstack aufgebaut werden, wie er in Smartphone Anwendungen typisch ist. Im Vergleich zu dem Angular Router, der eine webähnlichere Navigation bietet.

4.2 Typescript

Typescript ist eine Programmiersprache, die von Microsoft im Oktober 2012 veröffentlich wurde [10]. Heute steht Typescript unter der Apache 2.0 License [11]. Typescript ist ein Superset von Javascript, das bedeutet, dass jeder gültige Javascript Code auch gültiger Typescript Code ist. Der Name Typescript kommt davon, dass Typescript der dynamischen Programmiersprache Javascript ein optionales Typesystem hinzufügt. Um mit Typescript zu arbeiten, muss lediglich Typescript über den Node Package Manager (NPM) installiert werden. Daraufhin können Typescript-Source Dateien mit dem Typescript Compiler (TSC) transpiliert werden. Dabei wird der geschriebene Typescript Code in Javascript Code übersetzt. Neben dem Typesystem bietet Typescript im Vergleich zu Javascript noch viele weitere Features. Zum Beispiel ist es nun möglich, Interfaces zu definieren, die dann in Klassen implementiert werden können. Weiterhin können mit Typescript neue Javascript Features unterstützt werden, ohne auf eine Kompatibilität mit älteren Browsern verzichten zu müssen. Typescript kann dafür neue Syntax und Sprachfeatures übersetzen in Code, der in älteren Browsern, die noch nicht neue Sprachfeatures von Javascript durchgängig unterstützen, verstanden wird. Typescript kommt vorallem in Angular und Ionic zum Einsatz, die komplett selbst in Typescript geschrieben wurden. So ist es sinnvoll, auch das Projekt selbst in Typescript zu schreiben, auch wenn theoretischerweise die Möglichkeit bestände, alles auch in normalem Javascript zu schreiben. In Typescript können auch externe Bibliotheken verwendet werden, die nicht in Typescript geschrieben wurden. Dafür müssen sogenannte

.tsd-Dateien geladen werden, die eine mögliche Typisierung dieser Library beinhalten und es damit dem TSC ermöglichen die Library korrekt zu verwenden. Typescript ist mit all diesen Features eine ideale Sprache, um sinnvoll größere Projekte zu schreiben. Allerdings sollte darauf geachtet werden, dass die Entwicklungsumgebung Typescript unterstützt um alle Vorteile der statischen Typisierung zu nutzen. Dabei ist besonders die Code-Completion durch das Typensystem eine große Hilfe.

4.3 RxJS

Dieser Abschnitt befasst sich mit Reactive Extensions Library for Javascript (RxJS). RxJS ist eine Javascript Library, die es ermöglicht funktionalen reaktiven Programmiercode in Javascript oder Typescript zu schreiben. RxJS kommt in Angular sehr viel zur Anwendung, wodurch eine Beschäftigung mit RxJS Sinn macht, sobald man größere Angular Projekte schreiben möchte. RxJS steht unter der Apache 2.0 License [12]. RxJS implementiert das Pattern des Observables, welches kurz als eine Mischung von Iterator und Observer Pattern aufgefasst werden kann. Man kann einem Observable *subscriben* und erhält damit Zugriff auf die über die Zeit von dem Observable asynchron emittierten Werte. Dabei können die Observables durch viele verschiedene, durch die funktionale Programmierung bekannten, Operatoren bearbeitet werden, wie zum Beispiel `filter()` order `map()`. Das folgende Beispiel zeigt wie ein Observable von einem Array gebildet werden kann, durch eine funktionale Operation bearbeitet und zuletzt subscribed wird, um die Werte auf der Konsole auszugeben.

```
1 const arrayStream = Rx.Observable.fromArray([1,2,3]);
2 const doubleArrayStream = arrayStream.map(v => v*2);
3 doubleArrayStream.subscribe(v => console.log(v));
4 // console.log(2)
5 // console.log(4)
6 // console.log(6)
```

Listing 1: Beispiel für ein Observable und Subscription

Wichtig dabei zu verstehen ist, dass ein Observable aus fast allem erstellt werden kann. So kann auch aus dem klassischen asynchronen Javascript Callback oder Promise ein Observable erstellt werden, welches den Wert nextet, sobald das Javascript Promise resolved hat oder der Callback stattgefunden hat. Weiterhin ist es durch die Unterstützung von RxJS in Angular durch die Async Pipe möglich, ohne einen Subscribe Call in der Komponentenklasse durch die Async Pipe auf die Werte von Observables zuzugreifen, direkt im HTML Code einer Angular Komponente. Dies erleichtert den geschriebenen Code immens und ist zudem noch performant, da Angular nun selbst verwaltet, wann die Subscription zu dem Observable nicht mehr nötig ist und durch einen `unsubscribe()` Call beendet werden kann. Dadurch werden Memory-Leaks verhindert, die durch vergessene bestehende Subscriptions entstehen [13].

4.4 Angular

Dieses Kapitel befasst sich mit der Angular Plattform. Angular ist ein Open-Source Framework mit der MIT-Lizenz [14]. Angular ist in Typescript geschrieben. Es greift dabei auf moderne Javascript Features wie eine neue Klassensyntax und Decorators zurück, die dann wiederum durch Typescript auch älteren Browsern zugänglich gemacht werden können. Das Grundkonzept von Angular sind sogenannte Komponentenklassen. Diese bestehen aus einem HTML-Template, einer Typescript Klasse und den dazu benötigten Metadaten. Die Komponentenklasse definiert die Daten, die in dem HTML-Template dargestellt werden können, während das Template auf Interaktion reagieren kann und zum Beispiel bei einem Klick Methoden aufrufen kann. Für die HTML-Templates und das sogenannte Event Binding bietet Angular eine erweiterte HTML-Syntax, in denen Angular Komponenten geschrieben werden. Listing 4.4 zeigt ein Beispiel für eine beispielhafte Definition einer Angular Komponente.

```
1  @Component({
2    selector: "simple-example",
3    view: `<p (click)="logPlanet($event)">Hallo {{ planet
       ↪ }}<p>`
4  })
5  export class ExampleComponent {
6    planet: string = "Earth";
7
8    public logPlanet(event: Event): void {
9      console.log(`<p> has been clicked! Planet ${this.planet}
         ↪ is not amused!`, event);
10   }
11 }
```

Listing 2: Beispiel für eine einfache Angular Komponente

Die runden Klammern in dem Paragraph HTML-Tag `(click)` stellen dabei das Event Binding auf das Javascript Klick Event dar. Die doppelten geschweiften Klammern sind dabei das *Property-Binding*, welches durch die Instanzvariable `planet` des `ExampleComponent` gefüllt wird.

Eine Angular Anwendung besteht dann zumeist aus verschiedenen definierten Komponenten, die ineinander geschachtelt agieren und Daten austauschen und sich davon abhängig neu updaten oder *Property-Bindings* updaten. Weiterhin hat Angular das *ng-Module* System. Mit diesen Angular Modulen können Komponenten zusammengefasst werden und Service Klassen verfügbar gemacht werden. So können Anwendungen einzelne Teile in ein eigenes Modul zusammenfassen, interne Klassen verstecken und nur die nötigsten Klassen sichtbar machen. Neben den einzelnen Programmiermodellen stellt Angular auch schon fertige Module bereit. Dazu gehört zum Beispiel das HTTP-Modul in dem Klassen bereitgestellt werden, mit denen HTTP-Requests erleichtert werden. So kann mit Angular schnell eine Anwendung entwickelt werden, da gängige Probleme durch die fertigen Module bereits gelöst sind. Neben diesem Komponentenmo-

dell gibt es in Angular noch einen Dependency Injector, der in 4.4.1 näher beschrieben wird.

4.4.1 Dependency Injection

Ziel dieses Kapitels ist die Dependency Injection zu erklären und aufzuzeigen, wie sie in dem Prototyp zur Anwendung kam. Angular kommt mit einem Dependency Injection System. Man muss damit kein eigenes Dependency Injection System bauen, sondern kann auf das von Angular zurückgreifen. Grundsätzlich ist ein Gedanke bei der Dependency Injection, dass sobald ein Objekt Abhängigkeiten hat, wie zum Beispiel Konstruktorvariablen, diese Abhängigkeiten durch ein drittes Objekt injeziert werden. Das Objekt mit Abhängigkeiten ist nicht mehr damit beschäftigt die Abhängigkeiten zu bauen, sondern bekommt diese Objekte durch den Injector. So wird Code sehr leicht von Abhängigkeiten entkoppelt und lässt sich sehr gut testen. Da man dem Injector auch andere Objektinstanzen injezieren lassen kann, solange das Interface stimmt, kann man damit auch sehr leicht Teile vom Code zeitweise austauschen. So lassen sich gerade in Ionic Klassen, die auf native APIs zugreifen und damit das Debuggen im Browser nicht mehr zulassen, austauschen. Über den Angular Provider können nun für die Entwicklung Klassen ausgetauscht werden mit Mock-Klassen, die das gleiche Interface besitzen jedoch nur statisch Daten zurückliefern. Dies verschnellert und erleichtert die Entwicklung erheblich. In dem Prototyp kommt der Injector auch noch in einer besonderen Form im `TriggerContainer` zum Einsatz. Dabei wird in einer Art FactoryMethode `buildTrigger()` die verschiedenen implementierten TriggerKlassen erstellt. Da diese aber nicht mit der `Injectable()` annotiert werden können, müssen die Abhängigkeiten von Hand mit `injector.get(Dependency)` injeziert werden. So kann dynamisch je nach allen konfigurierten Triggern für den Patienten die passenden Trigger instantiiert werden [15].

4.5 Ionic Framework

Da der Prototyp mit dem Ionic Framework entwickelt wurde, wird in diesem Kapitel das Ionic Framework kurz vorgestellt. Das Open-Source Framework Ionic wird von Drifty Co. entwickelt [16]. Es steht unter der MIT License [17]. Ionic ist dabei, wie Angular auch, in Typescript geschrieben. Ionic bietet verschiedene Werkzeuge für den Entwickler. Zum einen bietet Ionic ein Command Line Interface (CLI) mit der Ionic Projekte erstellt, gebaut, deployed und debugged werden können. So wird Ionic auch über NPM installiert und kann von der Kommandozeile aus gesteuert werden. Ionic Apps können für verschiedene Plattformen gebaut werden, für den Prototypen wurde jedoch hauptsächlich auf die iOS Version geachtet. Neben der CLI besteht ein großer Teil von Ionic aus fertigen Angular User Interface (UI) Komponenten. Die Komponentenlibrary ist sehr groß und die meisten Interface Elemente aus nativen Apps sind darin zu finden. Darunter zum Beispiel Listen, Inputfelder, Tab-Layouts und Menus. Der Vorteil dieser Komponenten ist, dass sie sich an das jeweilige plattformspezifische Design anpassen. So sind zum Beispiel die Tabs unter iOS am unteren Rand des Screens, wandern jedoch automatisch in der Windows-Phone Variante nach oben. Neben den verschiedenen Angular Komponenten, bietet Ionic auch eine API für typische App Funktionalitäten. So bietet Ionic ein eigenes Konzept für Navigation zwischen einzelnen Screens mit dem `NavController`. Außerdem gibt es neben den UI Komponenten, einer API für hybride Applikationen, CLI auch ein Icon Set mit SVG Icons. Diese sind einfach verwendbar über Ionic Komponenten. Ein wichtiger Teil von Ionic ist jedoch die Kommunikation der API über Cordova und dem nativen OS. Dafür bietet Ionic auch noch die Ionic-Native Schnittstelle. Diese wird im nächsten Teil näher erklärt.

4.5.1 Ionic-Native

Wie bereits erwähnt in 4.5 bietet Ionic auch noch die Ionic Native Schnittstelle. Ionic-Native ist dabei eine Wrapper-Library um bestehende native Cordova Plugins, mit denen der Zugriff auf native Funktionen möglich ist. Ein Problem bei der Verwendung von einem Cordova Plugin ohne Ionic Native wäre, dass es anstatt Typescript lediglich Ja-

vascript verwendet [18]. Die meisten Cordova Plugins basieren außerdem auf Javascript Callbacks, mit dem reaktiven Ansatz mit Observables von RxJS ist aber eine API mit Promises und Observables sinnvoller. So können nämlich die kompletten RxJS Bibliotheksfunktionen auf die Daten der Plugins angewendet werden. Somit kann auch jetzt der Gedanke hinter Ionic-Native erklärt werden. Ionic-Native stellt um viele der bestehenden Javascript Cordova Plugins einen Wrapper, der die verschiedenen Callbacks und APIs der Plugins in eine API mit Observables, Promises und Typisierung für Typescript verfügbar macht. Ein Problem, welches bei der Entwicklung des Prototypen aufgetaucht ist, ist dass besonders die Ionic Plugins sehr fehleranfällig sind. So funktionierte das iBeacon Plugin nicht, außerdem gab es keinerlei Fehlermeldung über eine Fehlfunktion. Die Dokumentation der einzelnen Ionic-Native Wrapper ist oft unvollständig oder unterscheidet sich von der Dokumentation der ursprünglichen Cordova Plugins. Ähnlich wie das iBeacon Plugin funktionerte auch das Geofencing Plugin nicht. Grundsätzlich lässt sich sagen, dass es sehr schwierig und vorallem zeitaufwendig ist native Plugins zu debuggen. Sollte also ein natives Plugin nicht funktionieren, kostet es sehr viel Zeit den Fehler zu finden und erfordert auch ein erhebliches Wissen über native Programmierung mit der jeweiligen plattformspezifischen Sprache um den Fehler dann letzendlich auch zu beheben. Dies steht dann wieder im direkten Gegensatz zu dem Gedanken, sich mit der Entwicklung mit Ionic Zeit zu sparen oder ohne tieferes Wissen über die native App Programmierung native Funktionen zu implementieren.

4.6 Universally Unique Identifier

In diesem Abschnitt werden Universally Unique Identifier eingeführt und vorgestellt. UUIDs sind 128 Bit lange Zahlen die als Identifizierer dienen können. Es gibt verschiedene Typen von UUIDs, jedoch ist für die prototypische Implementation lediglich interressant, welche Vorteile eine Verwendung von UUIDs bringt. Für eine finale Version ist es wichtig, dass ein korrekt implementierter Algorithmus die UUIDs in der App erzeugt. Somit kann die Wahrscheinlichkeit für eine Kollision reduziert werden. Denn der theoretische Nachteil von UUIDs ist, dass sie gleiche UUIDs erzeugen könnten. Der dabei auftretende Effekt ist, dass Daten überschrieben werden könnten und damit

verloren gehen. Dies kann im Kontext von medizinischen Daten kritisch sein. Durch die konsequente Verwendung von UUIDs ist das Zusammenführen von Daten auf dem Server einfacher, da UUIDs im Vergleich zu aufsteigenden Identifizierern nur sehr unwahrscheinlich Kollisionen erzeugen [19].

5

Entwurf

Dieses Kapitel stellt den Entwurf der Anwendung dar. Dabei werden in einem Anwendungsfalldiagramm die unterschiedlichen Anwendungsfälle des Prototypen vorgestellt. In der Dialogstruktur wird der Ablauf der einzelnen Anwendungsbildschirme erläutert und deren mögliche Übergänge. Dabei ist auch ein möglicher Designentwurf zu sehen mit den entsprechenden Screenshots der Anwendung. Weiterhin wird passend zur Dialogstruktur dabei auch die einzelnen Pages und deren Funktionalität vorgestellt.

5.1 Anwendungsfalldiagramm

Dieses Kapitel beinhaltet das Anwendungsfalldiagramm 5.1, welches die verschiedenen Use-Cases der App wiederspiegelt.

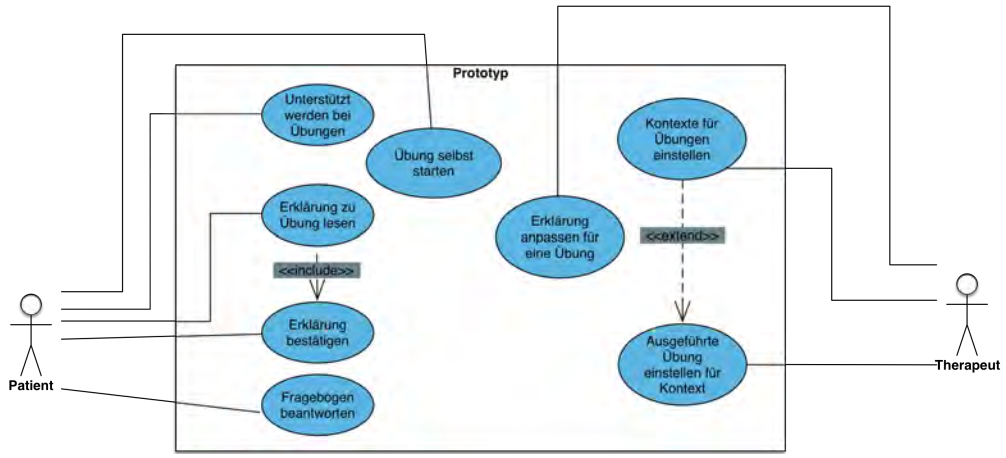

Abbildung 5.1: Anwendungsfalldiagramm

5.2 Dialogstruktur

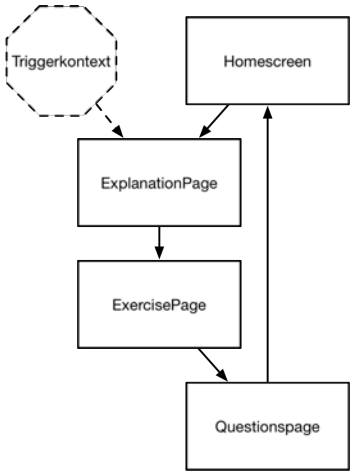

Abbildung 5.2: Dialogstruktur Diagramm

In den folgenden Kapiteln werden die Dialogstruktur und die einzelnen Pages des Prototypen genauer vorgestellt. Das Dialogstrukturdiagramm wird in 5.8 dargestellt. Das Achteck stellt dabei den Triggerkontext dar, der eine Exercise startet. Dies ist zum Beispiel für den Prototypen ein `ShortAfterAppLaunchTrigger`, der nach einer festgesetzten Zeit nach dem Start der App eine Übung ausführt.

Grundsätzlich ist der Prototyp einfach aufgebaut. Es gibt einen Homescreen, der Informationen zu Übungen darstellt und eine Möglichkeit bietet Exercises zu starten. Sobald eine Exercise durch einen Trigger gestartet wird, öffnet sich die Explanationpage. Diese enthält verschiedene Erklärungen zu der ausgelösten Exercise, durch verschiedene Erklärungselemente. Hiernach kann der Patient durch Buttonklick die Exercise starten. Hiernach wird die ExercisePage dargestellt, die nun den Patient bei der Ausführung unterstützt. Nach dem Ende der Übung wird übergeleitet auf die QuestionsPage. Hierbei

kann der Patient noch Fragen beantworten. Nach Bestätigen der Fragen setzt sich der Prototyp wieder auf den Homescreen zurück. Im Folgenden wird genauer auf die jeweiligen Pages eingegangen.

5.2.1 HomePage

Abbildung 5.3: Homescreen der Anwendung [20]

In der Abbildung 5.3 ist die Startseite des Prototypen zu sehen. Nachdem der App Splashscreen geladen hat, wird diese Page dargestellt. Direkt sichtbar auf dem Screen sind die Buttons, mit denen eine bestimmte Übung gestartet werden kann. Um möglichst auf den Patienten abgestimmt zu sein, können diese Buttons konfiguriert werden. So

kann der Gradientenfarbverlauf angepasst werden und das angezeigte SVG-Icon im Button kann verändert werden.

5.2.2 ExplanationPage

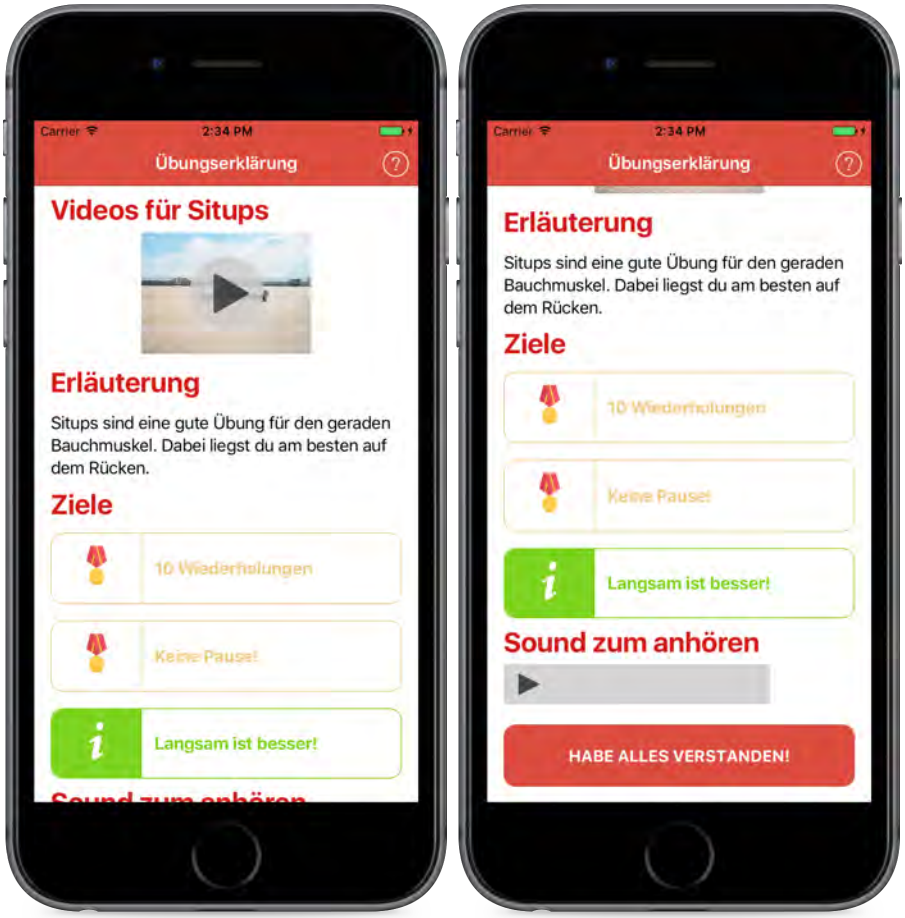

Abbildung 5.4: Übungserklärung [20] [21]

Ein wichtiger Teil des Prototypen war eine gute Erklärung für das Durchführen der Übung zu ermöglichen. Dafür existiert in dem Prototypen die ExplanationPage 5.4.

Es können verschiedene Arten von Erklärungen aneinandergereiht werden. Ebenfalls kann die Reihenfolge frei verändert werden, in denen die Erklärungen dargestellt werden. Es ist möglich auch Blöcke mehrfach zu verwenden. Dadurch ist eine sehr freie

Gestaltung der Erklärungen für den Therapeuten möglich. Die verschiedenen Erklärungsarten werden durch den Typescript Type `ExplanationBlockType` definiert und unterschieden.

```
1  export type ExplanationBlockType = "TEXT_ONLY" |
    ↪ "BADGE_LIST" | "VIDEO" | "AUDIO" ;
```

Listing 3: Die verschiedenen ExplanationBlockTypes

Im Folgenden werden die verschiedenen Arten von Erklärungen erläutert.

- TEXT_ONLY
  Der TEXT_ONLY ExplanationBlock ist ein einfacher Textblock mit Überschrift.
- BADGE_LIST
  Die BADGE_LIST ist eine Reihe von herausstechenden farbigen Kurzinformationen. Sie ist eine Möglichkeit für den Therapeuten, Ziele und Hinweise optisch abzusetzen. So kann der Therapeut Ziele und Kurzhinweise über die Übung mit einer Liste an Badges darstellen. Bisher unterstützt der Prototyp die zwei Badge Typen GOAL und INFORMATION. Diese werden durch den Typescript Type `BadgeType` identifiziert.

```
1  export type BadgeType = "GOAL" | "INFORMATION";
```

Listing 4: Der BadgeType

- VIDEO
  Der VIDEO Block ermöglicht es als Erklärung ein Video zu verwenden.
- AUDIO
  Ähnlich wie der VIDEO Block verwendet der AUDIO den Mediaplayer des Smartphones um eine Erklärung als Audiodatei abzuspielen.

5.2.3 ExercisePage

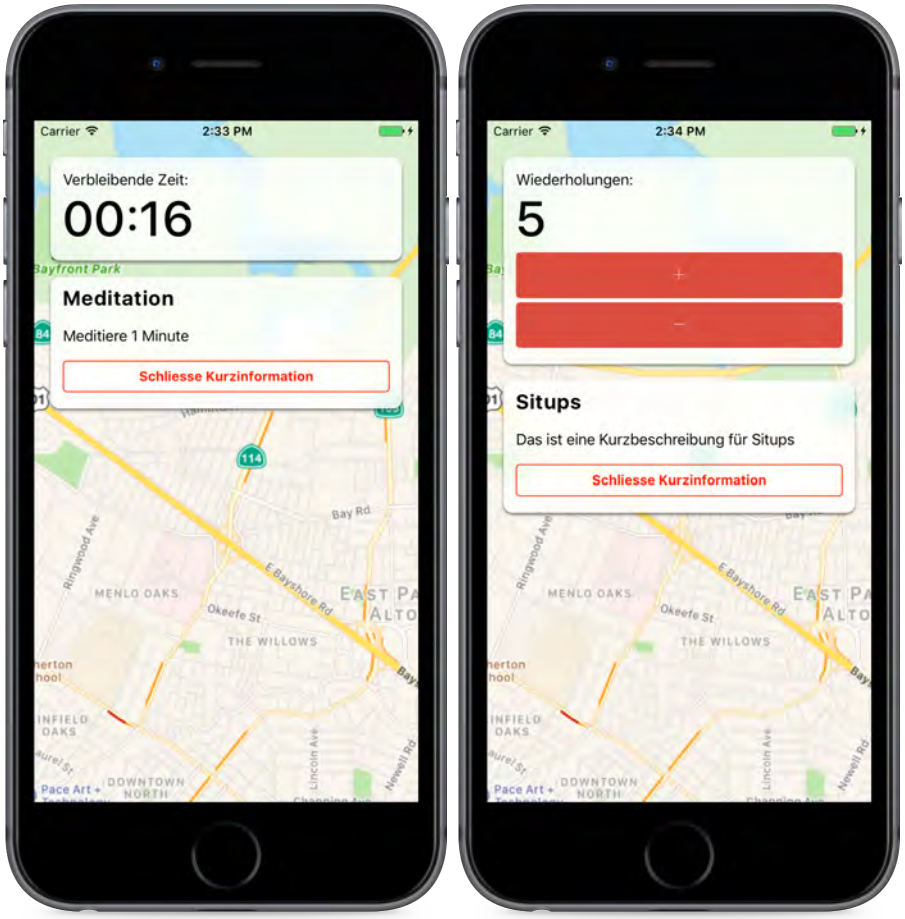

Abbildung 5.5: Die zwei Übungsscreens

Dieses Kapitel beschreibt die ExercisePage des App-Prototypen. Auf den Screenshots 5.5 sind die verschiedenen Übungsvarianten sichtbar. Gemeinsam haben beide Screenshots, dass die untere Box eine Kurzbeschreibung beinhaltet. Diese kann der Patient schliessen oder öffnen, um noch einmal die wichtigsten Informationen auch während der Übungsausführung zu besitzen. Der linke Screenshot des Prototypen zeigt eine Übung, bei der der Patient durch einen Counter unterstützt wird. Um die Übung erfolgreich abzuschließen, muss er eine bestimmte Anzahl an Repetitonen schaffen. Die Anzahl der Repetitonen kann durch die zwei Buttons erniedrigt und erhöht werden. Das

Wiederholungsziel kann konfiguriert werden. Diese Aufgabenart könnte zum Beispiel für das Zählen von Wiederholungen bei einer Kraftübung verwendet werden. Auf der rechten Seite ist ein Countdown sichtbar. Diese Übung ist beendet, sobald der Counter auf 0 fällt. Dabei kann die Dauer konfiguriert werden. Diese könnte zum Beispiel benutzt werden für eine Entspannungsübung, in der eine Minute ruhig geatmet werden soll. Eine finale Implementation könnte dabei den möglichen Übungsumfang erweitern. Das Design mit den verwischten Hintergrund und der einzelnen Karten kann dabei für weitere Übungen angepasst werden und lässt sich somit auch auf weitere neue Übungsarten anpassen. Durch das platzsparende Design, könnte der Hintergrund bei einer finalen Version dafür genutzt werden, dem Nutzer zum Beispiel über eine Karte die Route für seine Laufübung anzuzeigen. Weitere Bedienelemente können einfach in neuen Karten hinzugefügt werden.

5.2.4 QuestionsPage

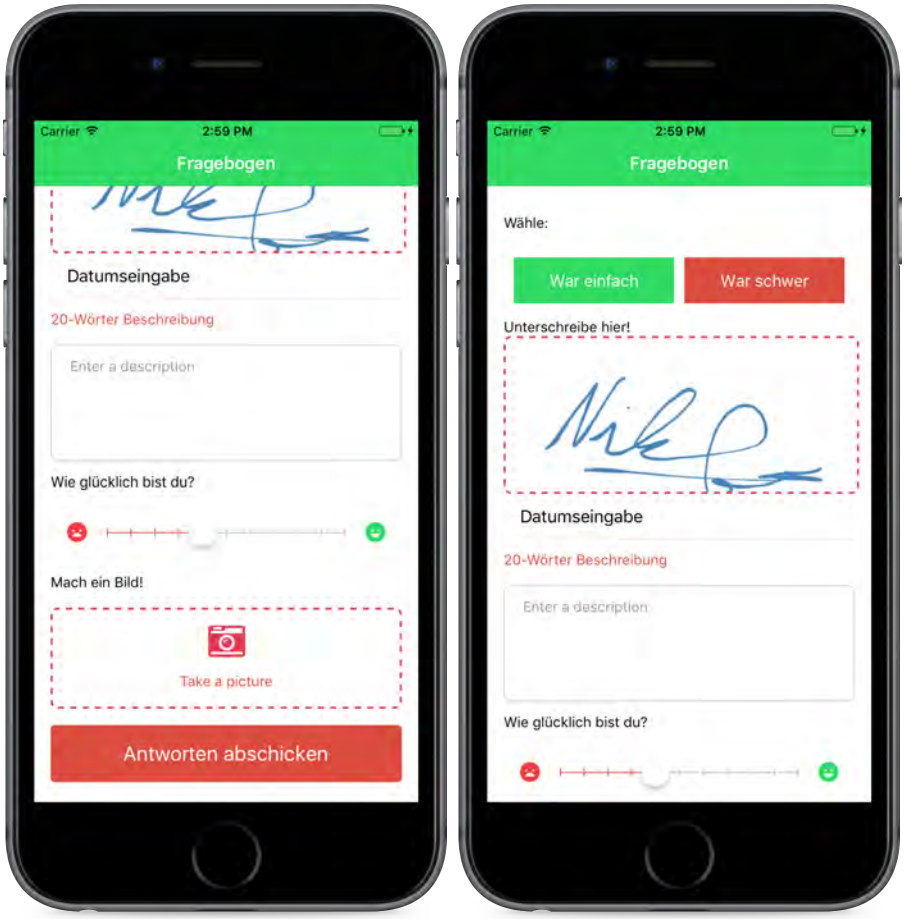

Abbildung 5.6: Fragebogen nach der Exercise

Dieser Abschnitt beschäftigt sich mit der Questionspage der Applikation. Nach dem erfolgreichen Absolvieren einer Übung, startet ein Fragebogen. In diesem Fragebogen kann der Patient nun verschiedene Fragen beantworten und damit die Übungsausführung komplett abschließen. Der Therapeut hat hierbei verschiedene Möglichkeiten den Fragebogen aufzubauen. Als Erstes ist ein Eingabefeld sichtbar, in dem der Patient zeichnen kann. Dies könnte zum Beispiel dafür verwendet werden um eine Unterschrift von dem Patienten zu verlangen. Es ist auch ein DateTime Eingabefeld sichtbar, mit dem sich eine DateTime Auswahlmethode öffnet. Neben einem normalen Texteingabefeld,

gibt es einen Stimmungsslider, der durch farbige Smileys dem Patienten die Möglichkeit gibt, die Stimmung nach der Übungsausführung zu erfassen. Auch kann der Patient mit der Bildfrage durch einen Tap auf das Kamerasymbol die Kamera des Smartphones öffnen und damit ein Bild schießen, welches dann für den Therapeuten sichtbar wird. So könnte zum Beispiel der Therapeut verlangen ein Bild von der bewegten Muskelpartie zu machen und mögliche Verletzungen über das Internet zu überwachen.

5.3 Klassendiagramm

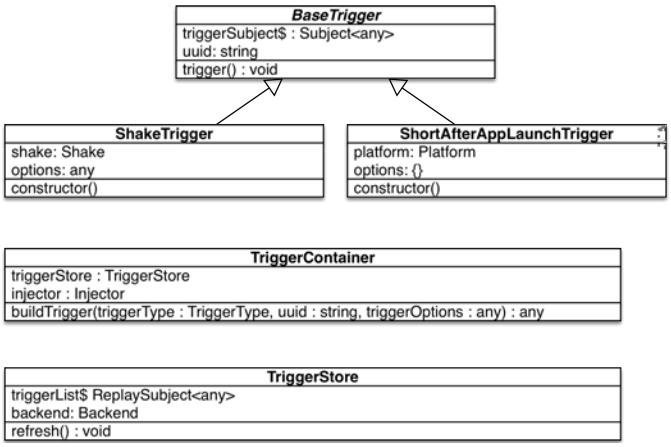

Abbildung 5.7: Klassendiagramm für das Triggerkonzept

Diese Kapitel stellt das Klassendiagramm vor für die Implementierung der kontextsensitiven Übungsausführung mit verschiedenen Triggern. Durch die Implementation mit der abstrakten Klasse `BaseTrigger` ist es problemlos möglich neue Trigger zu implementieren, die andere Kontexte für ein Auslösen benutzen. So können weitere Trigger und damit Kontexte beachtet werden, indem einfach neue Klassen implemtiert werden die von der `BaseTrigger` Klasse erben. Ein Trigger wird dann ausgelöst, sobald die `trigger()` Methode aufgerufen wird. Da die Konfiguration, welche Trigger welche Übungen ausführen voneinander unabhängig sind, kann so ein flexibles kontextbasiertes Auslösungsmodell für Übungen umgesetzt werden. Der `TriggerContainer` wiederum ist dafür verantwortlich, dass alle Trigger, die der Therapeut eingestellt hat instanziiert

werden. Dieser sucht aus dem TriggerMapping zu den einzelnen Triggern eine passende Übung. Dadurch, dass die Trigger jeweils eine UUID haben, kann also ein Trigger mit einer Übung gematched werden, indem einfach die UUIDs der Trigger und der Übungen verglichen werden aus der TriggerMap. Der TriggerStore ist dafür verantwortlich, die JSON-Konfiguration der Trigger zu aktualisieren beziehungsweise zu laden.

5.4 JSON-Strukturen

Dieses Kapitel erläutert die verwendeten JSON-Strukturen, die in dem Prototyp Anwendung finden. Als Erstes wird die Repräsentation einer Exercise erklärt. Die Exercise repräsentiert dabei eine Vorlage für eine Übungskonfiguration. Das Attribute `"uuid"` spezifiert dabei die Universally Unique Identifier (UUID) die eine Übungsausführungsvorlage identifiziert. Durch diese UUID lässt sich einfach zurückverfolgen, unter welcher Konfiguration eine Übungsausführung stattgefunden hat, unter der Vorraussetzung, dass jede neue Übungskonfiguration eine neue UUID zugewiesen bekommt. Das `"name"` Attribut ist ein einfacher Name der Übung. Das `"short_description"` Attribut ist eine kurze Beschreibung, die während der Übungsausführung auf dem Bildschirm 5.5 in einer zusammenklappbaren Beschreibungsbox sichtbar ist. Diese hilft dem Patienten, während der Übungsausführung die wichtigsten Punkte der Übung zu beachten. Das `"task_type"` Attribut bestimmt die Übungsart, die ausgeführt wird. Dabei ist in dem Prototypen ein `TaskType` definiert, der die unterschiedlichen Übungen angibt. So kann die Anwendung unterscheiden, welche Aufgabe der Patient ausführen muss. Die jeweiligen Konfigurationen für die Ausführung ist dann analog in dem `"task_options"` definiert. Da verschiedene Übungsarten verschiedene Ziele haben können, unter denen eine Übung als abgeschlossen gilt, sind in diesem Attribut für die jeweiligen Übungen die Bedingungen definiert, unter denen die Übung als erfolgreich abgeschlossen gilt. Unter dem Attribut `"explanation"` ist die für die Übung zugehörige Übungserklärung definiert. Wie in 5.2.2 erläutert, ist diese Liste eine Repräsentation der verschiedenen `ExplanationBlockTypes`. Zur individuellen Konfiguration des Homescreens kann in dem `"home"` Attribut der Farbverlauf der Buttons und ein Pfad zum eingestellten Icon

für die Übung angegeben werden. So kann der Homescreen auf die unterschiedlichen Übungen angepasst werden.

```
{
  "uuid": "44ea7a38-77f9-4ba2-82fa-5dc13792c2a7",
  "name": "Situps",
  "short_description": "Das ist eine Kurzbeschreibung fur
   ↪ Situps",
  "task_type": "ALBA_COUNTING",
  "task_options": {
    "goal": 10
  }
}
```

Listing 5: JSON-Konfiguration einer Exercise

Ähnlich wie zur Übungskonfiguration, kann auch für die verschiedenen Triggerkonfigurationen eine JSON Datenstruktur definiert werden. Dabei ist das Attribute `"uuid"` wieder eine eindeutige UUID, die den Trigger genau identifiziert. Das Attribut `"trigger_type"` spezifiziert analog wie der `"task_type"` den Typ des Triggers. 5.4 zeigt die unterschiedelichen Triggertypen. Dabei ist der `TriggerType` und das Konzept näher in 5.5 erklärt. In dem Attribute `"trigger_options"` der Inhalt des wieder unterschiedlich, da die unterschiedlichen Trigger auch alle unterschiedliche Konfigurationen benötigen. So hat der `ShakeTrigger` eine Sensitivät, die die Stärke des Schüttelns bestimmt oder der `ShortAfterAppLaunchTrigger` eine Zeitoption, die die Länge bestimmt, welche die Anwendung warten soll, bis eine Übung gestartet wird. Das Pedometer erwartet hier dann die Schrittanzahl, die für das triggern erreicht werden soll.

```
"explanation": [
  {
    "title": "Videos für Situps",
    "type": "VIDEO",
    "weight": "1000",
    "video_url": "http://www.google.de"
  },
  {
    "title": "Erläuterung",
    "type": "TEXT_ONLY",
    "weight": 100,
    "content": "Situps sind eine gute Übung für den geraden
     ↪ Bauchmuskel. Dabei liegst du am besten auf dem
     ↪ Rücken."
  },
  {
    "title": "Ziele",
    "type": "BADGE_LIST",
    "weight": 200,
    "badge_list": [
      {"text": "10 Wiederholungen", "type": "GOAL"},
      {"text": "Keine Pause!", "type": "GOAL"},
      {"text": "Langsam ist besser!", "type": "INFORMATION"}
    ]
  },
  {
    "title": "Sound zum anhören",
    "type": "AUDIO",
    "weight": 10,
    "audio_url": "http://www.google.de"
  }
]
```

Listing 6: JSON-Konfiguration einer ExerciseExplanation

```
    export type TriggerType =
    "ALBA_AFTER_APP_LAUNCH_TRIGGER" |
    "ALBA_SHAKE_TRIGGER" |
    "ALBA_PEDOMETER_TRIGGER"
```

Listing 7: Die verschiedenen Triggertypen

5.5 Triggerkonzept

Ein Hauptteil des Prototypen sollte aufzeigen, wie eine Architektur für eine dynamisches Auslösen von Übungen aussehen kann. Dabei wurde das Triggerkonzept entwickelt. Ein Trigger ist ein Objekt, welches auf einen beliebigen spezifizierten Zustand hört, Seiteneffekt auslöst und bekannt gibt, dass er ausgelöst wurde. Ein simples Beispiel ist der `ShortAfterAppLaunchTrigger`. Dieser triggert, je nach der konfigurierten Zeit in Millisekunden, nachdem die App gestartet wurde. Dazu muss diese Triggerklasse lediglich von der `BaseTrigger` Klasse erben und je nach gewollten Triggerlogik die `trigger()` Methode aufrufen. Allen Triggerklassen können verschiedene Optionen im Konstruktor mitgegeben werden, damit die Eigenschaften konfiguriert werden können. So ist es möglich, den gleichen Trigger öfters in der App zu verwenden, also zum Beispiel einer Minute und fünf Minuten nach dem Appstart den `ShortAfterAppLaunchTrigger` zu triggern. Außerdem ist das Hinzufügen von neuen Triggern damit sehr einfach und sehr schnell. Es muss lediglich eine reine Typescript Klasse erstellt werden, die von dem `BaseTrigger` erbt. Nun kann freie Logik implementiert werden, alles was geschehen muss ist, dass der Trigger irgendwo die `trigger()`-Methode aufruft. Ist der Trigger fertig implementiert, kann in der TriggerContainer Klasse in der `buildTrigger()`-Methode das Switch-Case Statement erweitert werden. Nun kann über die normale Triggerkonfigurationsfile die Optionen für den Trigger mitgegeben werden und die App instantiiert den neu hinzugefügten Trigger. Wichtig ist dabei anzumerken, dass der Trigger lediglich dafür verantwortlich ist, dass er irgendwann triggert. Er benötigt keine Informationen darüber, welche Übung er startet, dafür ist ein anderer Teil der Architektur verantwortlich. So müssen nicht die Trigger ausgetauscht werden, sollte eine andere Übung gestartet werden, sondern nur falls es neue Einstellungen geben sollte, wann ein Trigger auslösen soll. Für die weiteren Schritte nach dem Auslösen von Triggern ist das `TriggerMapping` verantwortlich, welches in den folgenden Kapiteln erläutert wird.

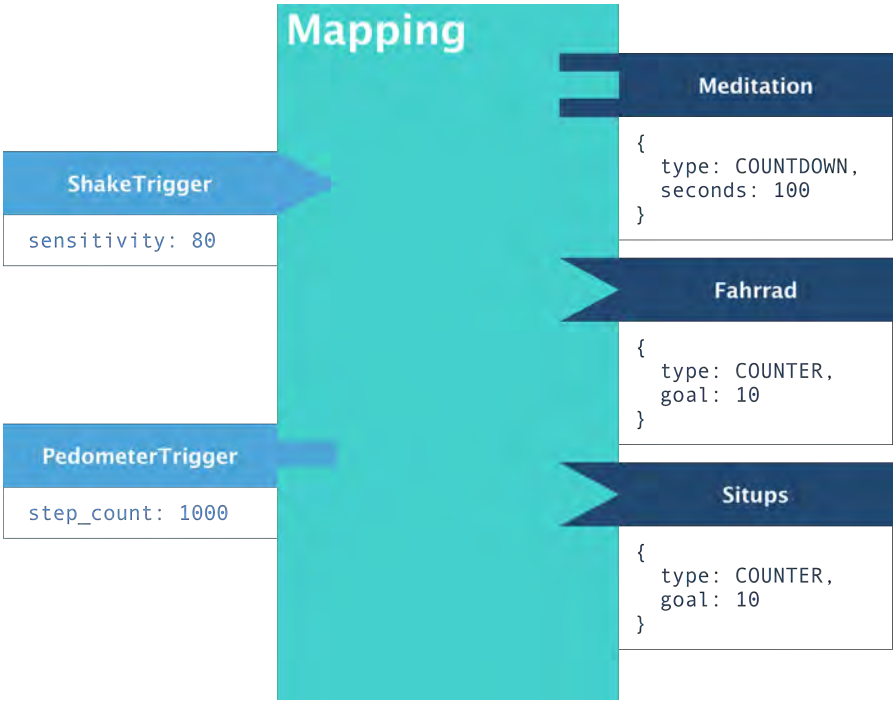

Abbildung 5.8: Das Triggerkonzept graphisch dargestellt.

5.6 Triggerbeispiele

Dieses Beispiel zeigt ein Beispiel für einen Trigger, wie er in der Anwendung implementiert wurde. Dieses Kapitel zeigt damit auf, wie einfach der Anwendung neue Trigger hinzugefügt werden können. Das darauffolgende Kapitel zeigt wie diese Trigger dann durch ein Triggermapping eine neue Übungsausführung starten. Der ShakeTrigger wird aktiviert durch das Schütteln des Smartphones. Dabei nehmen die Gyroskop Sensoren des Geräts das Schütteln war und lösen den Trigger aus. Dies geschieht über die `Shake` Klasse, die ein Ionic Native Plugin ist. Die `startWatch(sensitivity)` Methode übergibt ein Observable, welches emittiert, sobald das Smartphone, über die als Parameter übergebene Sensitivität, geschüttelt wird. In dem erfolgreichem Callback des Observables wird einfach nur die `trigger()` Methode des `BaseTrigger` aufgerufen. Dadurch wird der Anwendung signalisiert, dass der Trigger getriggert hat. Über den Parameter `options` kann ein Trigger entsprechend konfiguriert werden. So kann

beispielsweise bei dem ShakeTrigger die Sensitivität, auf die der Trigger reagieren soll, eingestellt werden. Weiterhin übergibt der `ShakeTrigger` das von dem `BaseTrigger` geerbte `TriggerSubject` als reines Observable über eine Typescript Getter Methode. Das Prinzip hinter einem Trigger ist sozusagen immer das Gleiche. Es ist eine normale Typescript Klasse, die von der `BaseTrigger` Klasse erbt. Der Trigger benötigt dann eine UUID, die entsprechenden Ionic-Native Klassen, sowie ein Options Parameter. Nun kann beliebiger Anwendungscode geschrieben werden, sobald der Trigger auslösen soll, wird die `trigger()` Methode aufgerufen. Ein Vorteil dieser Herangehensweise ist, dass ein Trigger lediglich dafür verantwortlich ist, zu Triggern. Ein Trigger weiß nichts darüber, welche Übungsausführung er starten soll. So kann ein Kontext unabhängig davon gestaltet werden, welche Übungen existieren. Die Applikation unterstützt prototypisch auch noch weitere Trigger, darunter zum Beispiel ein PedoMeter, welches nach einer bestimmten Schrittzahl triggert. Weiterhin auch ein Trigger, der nach einer bestimmten Zeit nach Starten der Anwendung triggert. Da dieses Konzept sehr allgemein verwendbar ist, wäre denkbar auch weitere Geräte bei Triggern einzubinden. So könnte ein Trigger programmiert werden, der über eine Pulsuhr den Puls misst und ab einer spezifizierten Pulsgrenze triggert.

5.7 Triggermapping

Wie in 5.6 bereits erwähnt, ist nicht der Trigger dafür verantwortlich zu wissen, welche Übung er ausführt. Für das Bestimmen der auszuführenden Übung ist das TriggerMapping zuständig. Dabei gibt es ein `"trigger_uuid"` Attribut und ein `target_exercise_uuid` Attribut. Sobald ein Trigger auslöst, wird in der `AppComponent` über den `MapConfigStore` und dessen `getByTriggerUUID(uuid)` Methode das passende Mapping mit der UUID des Triggers und die auszuführende Exercise herausgesucht. Sobald das passende Mapping gefunden wurde, sucht der `ExerciseStore` die Übungsvorlage heraus, die dann durch wechseln auf die ExplanationPage gestartet wird.

6

Anforderungsabgleich

Dieses Kapitel gleicht die Anforderungen mit dem Prototypen ab. So sind nicht alle Anforderungen möglich perfekt mit Ionic umzusetzen, wiederum manche Aspekte dabei sehr gut gelungen. Die erste funktionale Anforderung, das Starten von Übungen F1, funktioniert in dem Prototypenk sehr gut. So startet eine Übung wie in dem Kapitel 5.5 beschrieben. Dabei ist die Konfiguration von Übung und Kontext, unter dem eine Übung ausgelöst wird getrennt. Dies stellt einen großen Teil der Arbeit dar und verdeutlicht, dass es möglich ist ein dynamisches System zu bauen. Die nächste Anforderung, das Nutzen von Gerätesensoren F2 ist auch erfüllt worden. So können die Trigger moderne Funktionen wie zum Beispiel das Erkennen von Schütteln des Gerätes oder ein Schrittzähler verwenden. Dies funktioniert dank der Cordova Plugins auch sehr gut, jedoch gibt es einige Sensoren, die nicht angesteuert werden konnten, da die entsprechenden Cordova Plugins nicht funktionierten. Das Konfigurieren von Übungen F3 funktioniert über die erklärte JSON-Strukur und des Trigger Mappings 5.7 sehr dynamisch. Eine wichtige Anforderung war, dass Übungen über eine sinnvolle Erklärung erläutert werden F4. Die vielen verschiedenen Möglichkeiten für die Erklärungsblöcke, die in 5.2.2 näher beschrieben werden, erlauben es Therapeuten eine gründliche Erklärung für neue Übungen zu verfassen. Diese können auch moderne Mediaelemente nutzen, wie zum Beispiel eine Videoerklärung, somit wird auch Anforderung F6 erfolgreich abgedeckt. Bevor eine Übung gestartet werden kann, muss diese in der ExplanationPage erst bestätigt werden. Damit ist Anforderung F7 auch erfüllt. Die Anforderung eine iBeacon basierte Ausführung einer Übung zu starten, konnte nicht umgesetzt werden. So hat das Cordova Plugin, welches für die Verbindung mit iBeacons benötigt wird, nicht funktioniert. Auch ist ein Fehlersuchen ohne Kenntnisse in der nativen Programmierung in iOS nicht möglich und

damit das Beheben des Problems, ohne erheblichen Aufwand, nicht möglich gewesen. Hierbei zeigt sich auch wieder einmal die Abhängigkeit von den Cordova Plugins und deren Fehlerfreiheit. Durch die Triggerimplementation kann jedoch für einen weiteren Prototypen das Plugin repariert werden und ein neuer Trigger implementiert werden. Dies funktioniert analog zu allen anderen Triggern und zeigt, wie das Triggerkonzept leicht erweiterbar ist auf neue Sensoren und Funktionalitäten. Die Anforderung, dass eine Übungsausführung aus einem Repetitionzähler besteht, konnte implementiert werden F9. So kann der Patient die Wiederholungszahl während der Übungsausführung mitzählen und bei Erreichen des Ziels ist die Übungsausführung beendet. Ein Fragebogen der nach der Übungsausführung angezeigt wird besitzt der Prototyp ebenfalls F10. Dieser besitzt auch innovative Eingabefelder, wie zum Beispiel einen Stimmungsslider F12, Kameraantwort F13 und Grafikantwort F11. Damit werden die innovativen Felder abgedeckt. Jedoch funktioniert in dem Prototpyen nicht, dass diese ebenfalls dynamisch veränderbar sind. Der Prototyp zeigt damit nur, dass es theoretisch möglich ist, dass nach einer Ausführung Fragen angezeigt werden und verdeutlicht, wie ein solcher Fragebogen aussehen könnte. Der Bereich der nicht-funktionalen Anforderungen ist ebenfalls abgedeckt. So ist in der ersten nicht-funktionalen Anforderung Erweiterbarkeit NF1 ein erweiterbares System gefordert. Wie in 5.5 erläutert, ist das Kontext und Triggersystem sehr leicht erweiterbar. Auch die Konfiguration zur Laufzeit wie in F3 ist damit möglich. Neben reiner Funktionalität, wurde bei dem Prototypen auch Wert auf das Interface NF3 und Usability gelegt NF4. So wurden nicht nur standard Ionic-Komponenten verwendet, sondern diese abgewandelt und mit eigenem Cascading Stylesheets (CSS) geschrieben. Auch der Einsatz von Farben, wie zum Beispiel bei den Badges in der Erklärung vereinfacht das Auffassen von Informationen immens. Daher wurden auch diese Anforderungen erfüllt. Da die Anwendung bisher in Ionic wegen eines Fehlers noch nicht im Produktionsmodus gebaut wurde, jedoch eine akzeptable Performanz ohne Wartezeiten vermittelt, ist davon auszugehen, dass eine Version im Produktionsmodus auch die benötigte Performanz aufweist. Auch eine Offline-Verfügbarkeit NF6 ist möglich, da neben den verschiedenen Konfigurationen für das Starten oder bearbeiten der Übungen keine Internetverbindung bestehen muss.

Grundsätzlich lassen sich mit Ionic die meisten Anforderungen abdecken. Jedoch ist ein immer wieder aufkommendes Problem, dass viele Fehler oft im Detail versteckt sind. So ist Ionic und Angular ein extrem schnell sich entwickelndes und noch junge Frameworks, was wiederum bedeutet, dass viele APIs sich sehr schnell verändern. Das bedeutet auf der einen Seite, dass viele neue Features hinzugefügt werden, dabei aber auch immer wieder Teile des Frameworks Fehler aufweisen. So kommt es zu Grafikfehlern, die je nach Version von Ionic ohne Gründe wiederrum verschwinden oder Änderungen in Angular, die durch Änderungen im Framework dazu zwingen, Teile der Applikation upzudaten. Auch in Ionic-Native ist das Problem, dass sich die nativen Cordova Plugins ändern, jedoch der Ionic-Native Wrapper dadurch nicht automatisch geupdatet wird. So gibt es Plugins, die bereits eine neuere API bieten, als der Ionic-Native Wrapper und dessen Typisierung andeutet.

Anforderungsabgleichtabelle

Anforderung	Erfüllt	Begründung
F1	Ja	Starten von Übungen ist möglich.
F2	Ja	Es wird ein Schüttelsensor und Pedometer genutzt.
F3	Ja	Übungen werden über JSON konfiguriert.
F4	Ja	Erklärungen werden über JSON konfiguriert.
F5	Ja	Selbstauslösende werden über JSON konfiguriert.
F6	Ja	Es existieren Video- und Audioerklärungen.
F7	Ja	Erklärungen müssen bestätigt werden.
F8	Nein	iBeacon Cordova Plugin hat keine Reaktion gezeigt.
F9	Ja	Repetitionszähler ist eine Übungsart.
F10	Ja	Ein Fragebogen wird angezeigt nach Übungsausführung.
F11	Ja	Eine Zeichenfläche ist eine Antwortmöglichkeit.
F12	Ja	Einen Stimmungsslider ist eine Antwortmöglichkeit.
F13	Ja	Es können Antworten per Kamera gegeben werden.
NF1	Ja	Das Triggerkonzept erlaubt einfache Erweiterungen.
NF2	Ja	Durch JSON Konfigurationen ist alles auch veränderbar.
NF3	Ja	Das Interface besitzt extra entwickelte Komponenten.
NF4	Ja	Die eigenen Komponenten erlauben gute Usability.
NF5	0	Performanz kann durch Produktionsmodus gesteigert werden.
NF6	Ja	Internet ist nach laden von JSON nicht mehr nötig.

Tabelle 6.1: Anforderungsabgleichtabelle

7

Zusammenfassung

Bei der Konzeption und Realisierung einer kontextsensitiven mobilen Anwendung für therapeutische Hausaufgaben mussten verschiedene Aspekte bedacht werden. Dazu müssen die wissenschaftlichen Grundlagen therapeutischer Hausaufgaben verstanden werden, um die Bedürfnisse für eine sinnvolle Anwendung abzudecken, die für moderne Behandlungen wichtig sind. Eine sinnvolle Definition für Hausaufgaben ist dabei, dass sie zeitlich zwischen Therapiesitzungen stattfindet, die Aufgabenart jedoch sehr variabel ausfallen kann [2]. Da durch moderne mobile Endgeräte die Behandlung mit therapeutischen Hausaufgaben noch weiter verbessert werden kann, durch zum Beispiel effektivere Zeitnutzung in den Therapiesitzungen und eine verbesserte Datengrundlage für die Wissenschaft, ist es von Nöten, dass eine moderne Anwendung all die verschiedenen Aspekte ermöglicht [5]. Mit all diesen Informationen wurde eine Liste an Anforderungen für einen Prototypen entwickelt, der prototypisch eine mobile Anwendung in Teilen darstellt, die ein Patient und Therapeut für eine moderne Behandlung verwenden kann. Dabei sind konfigurierbare Übungen mit Erkärungen, das Einstellen von Kontexten, verschiedene Übungsarten und ein Fragebogen mit innovativen Eingabemöglichkeiten sinnvoll. Für die Programmierung des Prototypen wurde dabei eine hybride Ionic-Angular Anwendung in Typescript geschrieben, die über Apache Cordova auf native Funktionen des OSes Zugriff hat. Dabei wurde RxJS verwendet für die Verdeutlichung einer mögliche Architektur für ein Triggerkonzept unter dem die über JSON-konfigurierbaren Übungen näher erläutert. In einem Anforderungsabgleich werden zum Schluss die geplanten Anforderungen mit der tatsächlichen Umsetzung abgeglichen und auf Probleme der einzelnen Anforderungen auch in Hinblick der Frameworkwahl besprochen, wie das Nicht-Funktionieren verschiedener nativen Cordova Plugins.

7.1 Fazit

Abschließend ist festzuhalten, dass es auf jedenfall sinnvoll ist, eine Anwendung weiterzuentwickeln, die die verschiedenen Aspekte für therapeutische Hausaufgaben alle zusammenfasst. So hat Kapitel 2 deutlich gezeigt, wie sinnvoll therapeutische Hausaufgaben an sich in der Behandlung sind, aber auch wie mobile Geräte die Effektivität dieser steigern können. Der entwickelte Prototyp hat auch eine mögliches User-Interface dargestellt, welches für die Übungsausführung übernommen werden könnte. Auch das Triggerkonzept ermöglicht, dass sehr variabel der Kontext, der einzelnen Situationen des Patienten, erfasst werden kann. Bei der Technologiewahl sollte jedoch darauf Acht gegeben werden, dass für eine Anwendung, bei der der Fokus der Zugriff auf Sensorik und native APIs ist, hybride Applikationen immer Probleme bereiten, die sich nur mit überproportional viel Arbeitsaufwand nur teilweise beheben lassen. Dies entsteht vorallem durch die vielen Zwischenschichten, auf denen solche Anwendungen aufgebaut sind. Da aber der Prototyp an den meisten Stellen mit RxJS eine funktionierende Implementation gezeigt hat, lässt sich bei der Entwicklung des finales Programms viel der Logik übernehmen. So ist das Konzept von Observables und deren funktionalen Methoden und Kombinationen nicht abhängig von der Plattform sondern ein übergreifendes Konzept. Nach eigenem Empfinden ist Ionic auch ein sehr gutes Tool, um eine funktionale App zu bauen, an der Bedienkonzepte und eine prototypische Anwendung getestet werden kann. Ich sehe auch weniger das Problem in der Performance von hybriden Apps bzw. von Ionic, sondern bei der Verwendung der Cordova Plugins. Diese sind oft sehr unterschiedlich gut dokumentiert, stark auf gewisse OS Versionen limitiert und haben auch oft nur einen Teil der nativen API implementiert. Gerade dann, wenn die App viele verschiedene Sensoren verwenden soll, ist man dadurch sehr stark abhängig von den Libraries. Der Prototyp verdeutlicht, dass es auf jedenfall möglich ist, eine Implementation zu haben, die von Therapeuten geupdatet werden kann ohne die Apps zu updaten oder eine neue App zu entwickeln. Für mich besonders eindrucksvoll ist, dass eine gute generische Implementation mit den verschiedenen Aspekten des Albatross Projekts, realistisch vielen Patienten und Therapeuten zu einer viel besseren Behandlung führen könnte.

7.2 Ausblick

Neben der wachsenden Anzahl und dem erhöhten Bedarf von modernen Therapieoptionen bei dem Stellen von therapeutischen Hausaufgaben stellen, sich gerade moderne mobile Endgeräte als hilfreich dar. Da zurzeit noch keine generelle Lösung gefunden wurde, stellt sich die Frage, wann es ein System gibt, mit dem die benötigten Funktionalitäten an eine App, die unterstützend im Therapieverlauf sein soll, veröffentlicht wird. Wie der Prototyp gezeigt hat, ist es möglich eine App zu bauen, die mit entsprechendem Programmieraufwand ein breites Spektrum abdeckt. Daher ist gerade für die Zukunft interressant, den Prototypen zu nutzen und eine App zu entwickeln, die neben dem von diesem Prototypen gezeigtem Triggermodell auch weitere Funktionalitäten implementiert. So könnte die Übungsanzahl erweitert werden, Notificationsupport für die Applikation eingebaut werden, und einen konfigurierbaren Fragebogen die App sehr nützlich machen. Auch eine serverseitige Implementation, die jetzt die gesammelten Daten der App für den Therapeuten sinnvoll darstellt und einen guten Aufgaben- und Kontexteditor, mit dem der Therapeut die Aufgaben einstellen kann ist denkbar. Hoffentlich ist es nur noch eine Frage der Zeit, wie lange ein noch sinnvolles System entwickelt werden muss, da eine gut gebaute Anwendung Behandlungen enorm verbessern würde. Für eine sinnvolle Umsetzung sollte jedoch gerade für das Sammeln von Sensordaten und Kontextunterstützung ein nativer Entwicklungsansatz gewählt werden, da die Abhängigkeit von den Third-Party-Libraries ein zu hohes Risiko für den Erfolg des Projekts darstellt.

Abkürzungen

CLI Command Line Interface

UI User Interface

CSS Cascading Stylesheets

UUID Universally Unique Identifier

NPM Node Package Manager

TSC Typescript Compiler

RxJS Reactive Extensions Library for Javascript

Literaturverzeichnis

[1] Schickler, M., Pryss, R., Schobel, J., Reichert, M.: Supporting Remote Therapeutic Interventions with Mobile Processes. In: 6th IEEE International Conference on AI & Mobile Services (IEEE AIMS 2017), IEEE Computer Society Press (2017)

[2] Fehm, L., Fehm-Wolfsdorf, G.: Lehrbuch der Verhaltenstherapie - Therapeutische Hausaufgaben. (2009)

[3] Kazantzis, N., Whittington, C., Dattilio, F.: Meta-Analysis of Homework Effects in Cognitive and Behavioural Therapy: A Replication and Extension (2000)

[4] Schickler, M., Pryss, R., Schobel, J., Schlee, W., Probst, T., Reichert, M.: Towards Flexible Remote Therapeutic Interventions. In: 30th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2017), IEEE Computer Society Press (2017)

[5] Schickler, M., Pryss, R., Stach, M., Schobel, J., Schlee, W., Probst, T., Langguth, B., Reichert, M.: An IT Platform Enabling Remote Therapeutic Interventions. In: 30th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2017), IEEE Computer Society Press (2017)

[6] Apache: Apache Projects List. (http://www.apache.org/index.html#projects-list) [Zuletzt besucht: 14.11.2017].

[7] Cordova, A.: Architectural overview of Cordova platform - Apache Cordova. (https://cordova.apache.org/docs/en/7.x/guide/overview/index.html) [Zuletzt besucht: 01.12.2017].

[8] Google: Angular - Routing & Navigation. (https://angular.io/guide/router) [Zuletzt besucht: 13.11.2017].

[9] Ionic: NavController - Ionic API Documentation. (https://ionicframework.com/docs/api/navigation/NavController/) [Zuletzt besucht: 13.11.2017].

[10] Turner, J.: Announcing TypeScript 1.0. (https://blogs.msdn.microsoft.com/typescript/2014/04/02/
[Zuletzt besucht: 01.11.2017].

[11] Microsoft: Github Typescript License. (https://github.com/Microsoft/TypeScript/blob/master/LICENSE.txt)
[Zuletzt besucht: 01.11.2017].

[12] ReactiveX: RxJS License. (https://github.com/ReactiveX/rxjs/blob/master/LICENSE.txt)
[Zuletzt besucht: 02.11.2017].

[13] Google: AsyncPipe. (https://angular.io/api/common/AsyncPipe)
[Zuletzt besucht: 03.11.2017].

[14] Angular: Angular License. (https://github.com/angular/angular/blob/master/LICENSE)
[Zuletzt besucht: 14.11.2017].

[15] Google: Angular - The Dependency Injection pattern. (https://angular.io/guide/dependency-injection-pattern)
[Zuletzt besucht: 10.12.2017].

[16] Ionic: Ionic Framework - About. (https://ionicframework.com/about)
[Zuletzt besucht: 01.11.2017].

[17] Ionic: License Ionic Framework. (https://github.com/ionic-team/ionic/blob/master/LICENSE) [Zuletzt besucht: 04.11.2017].

[18] Ionic: Ionic Native. (https://ionicframework.com/docs/native/)
[Zuletzt besucht: 01.11.2017].

[19] Group, T.O.: DCE 1.1: Remote Procedure Call - Universal Unique Identifier. (http://pubs.opengroup.org/onlinepubs/9629399/apdxa.htm)
[Zuletzt besucht: 10.12.2017].

[20] Golubev, N.: SVG-Icons. (https://www.flaticon.com/authors/nikita-golubev) designed by Nikita Golubev from Flaticon, [Zuletzt besucht: 10.12.2017].

[21] Bruce, D.: SVG-Info-Icon. (https://www.flaticon.com/authors/daniel-bruce) designed by Daniel Bruce from Flaticon, [Zuletzt besucht: 10.12.2017].

A

Quelltexte

```
export class ShakeTrigger extends BaseTrigger {
  constructor(
    public shake: Shake,
    public uuid: string,
    private options: any
  ) {
    super(uuid);
    this.shake.startWatch(options['sensitivity']).subscribe(()
     ↪ => {
      this.trigger();
    });

    this.uuid = uuid;
  }

  get trigger$(): Observable<any> {
    return this.triggerSubject$.asObservable();
  }
}
```

Listing 8: ShakeTrigger Implementation

```
{
    "uuid": "3e9f15ff-4f91-4fec-a3f0-c3b74825cea6",
    "trigger_type": "ALBA_SHAKE_TRIGGER",
    "trigger_options": {
      "sensitivity": 80
    },
}
```

Listing 9: ShakeTrigger Konfiguration

```
[
  {
    "trigger_uuid": "3e9f15ee-4f91-4fec-a3f0-c3b74825cea6",
    "target_exercise_uuid":
     ↪ "798c1658-694a-4f8f-84a3-f7d935f0e26b"
  },
  {
    "trigger_uuid": "3e9f15ff-4f91-4fec-a3f0-c3b74825cea6",
    "target_exercise_uuid":
     ↪ "44ea7a38-77f9-4ba2-82fa-5dc13792c2a7"
  }
]
```

Listing 10: TriggerMapping

```
[
  {
    "uuid": "3e9f15ff-4f91-4fec-a3f0-c3b74825cea6",
    "trigger_type": "ALBA_SHAKE_TRIGGER",
    "trigger_options": {
      "sensitivity": 80
    }
  },
  {
    "uuid": "3e9f15ee-4f91-4fec-a3f0-c3b74825cea6",
    "trigger_type": "ALBA_AFTER_APP_LAUNCH_TRIGGER",
    "trigger_options": {
      "time": 5000
    }
  }
]
```

Listing 11: JSON-Konfiguration der Trigger

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Niklas Häusele Matrikelnummer: 781855

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Niklas Häusele